Attorney Docket No.: 5577-270

METHODS, SYSTEMS AND COMPUTER PROGRAM PRODUCTS FOR CONTROLLING CACHING OF DISTRIBUTED DATA

Field of the Invention

The present invention relates to distributed data and more particularly to the caching of distributed data.

Background of the Invention

Caching is a common technique used in applications running on application servers to gain performance and scale while reducing cost of ownership. Caching is well suited for the typical transactional web-based applications because of their high read to write ratios. For example, in an on-line trading application, much more time is spent watching (reading) the prices of stock versus buying or selling stock (writing). When caching is productively employed within an eBusiness application, traffic to costly backend servers, such as database servers or legacy code running on mainframes, may be reduced. Thus, less capacity of such backend servers may be required which may lower the total cost of owning the system.

High volume web applications typically use large clusters of application servers to scale to the response time demands of customers. Caching in large clustered environments can present challenges in keeping cached data consistent. One common method employed by application servers is illustrated in **Figure 1**. As seen in **Figure 1**, the application servers **10** have corresponding caches **12**. Message oriented middleware (MOM) **14** is utilized to replicate cached data and/or

data invalidation notification between application server instances **10** in the cluster of application servers. Thus, data from the database **16** may be replicated in the caches **12** of the application servers **10**. This replication may be controlled by the MOM **14** utilizing a message network.

Data replication in the caches **12** may be cost effective if the cost to replicate the data, which is usually measured in latency, is known to be more efficient than to recreate the data from the data source of origin. For example, if it takes 250ms to completely replicate an object containing a companies net worth and it takes 2500ms to calculate the companies net worth by interacting with backend data management systems, is may be preferable to replicate the net worth object rather than recalculate it. However, during peak times, when application servers are very busy, the busy CPUs and congested network may lead to delays which might cause the total replication of the net worth object to exceed 2500ms. Typically, however, the decision of whether to replicate (cache) the net worth object or to recalculate it is a static decision and does not depend on a particular situation for replication. Thus, typically, the net worth object would be cached or recalculated independent of the particular timings for replication and/or recalculation.

Another variation of cache distribution is illustrated in **Figure 2**. **Figure 2** illustrates off-loading of cache memory data to a disk cache. Thus, as seen in **Figure 2**, the caches **12** of the application servers **10** may write cache data to a file system **20** to offload the data to a disk cache. Off-loading cache data to disk may be an effective way of ensuring that the memory-based cache does not exhaust memory. Disk offload can also be used to distribute cache data between the servers **10** in the cluster of application servers if the disk file is stored on a networked file system. As with replication described above, typically, the decision of whether to offload cache data to disk is a static decision and does not depend on a particular situation for data offload.

Summary of the Invention

Embodiments of the present invention provide for controlling a cache of distributed data by dynamically determining whether and/or where to cache the distributed data based on characteristics of the data, characteristics of the source of

the data and characteristics of the cache so as to provide an indication of whether and/or where to cache the data. The data may be selectively cached based on the indication. Moreover, the data ma be cached to memory, disk, replicated within a cluster, and/or cached in other devices/locations, based on the indication.

In particular embodiments of the present invention, the characteristics of the data include how often the data is accessed. The characteristics of the source of the data may include how long it takes to recompute the data and/or how long it takes to replicate the data. The characteristics of the cache may include how long it takes to retrieve a cached item.

In still further embodiments of the present invention, dynamically determining whether and/or where to cache the distributed data includes determining a predicted maximum number of cache accesses, determining a predicted maximum time consumed by processing cache hits corresponding to a cache entry corresponding to the distributed data, determining a time (r) to replicate the distributed data and determining time (c) to generate the distributed data. The indication is set to indicate caching the distributed data if the sum of the time to generate the distributed data, the time to replicate the distributed data and the predicted maximum time consumed by processing cache hits is less than the product of the predicted maximum number of cache accesses and the time to generate the distributed data.

Additionally, a time to live (TTL) for the cache entry corresponding to the distributed data, a time (h) to process a cache hit corresponding to the distributed data and a predicted frequency (f) of cache accesses for the cache entry corresponding to the distributed data may also be determined. In such cases, determining a predicted maximum number of cache access may be provided by determining TTL*f. Also, determining a predicted maximum time consumed by processing cache hits corresponding to a cache entry corresponding to the distributed data may be provided by determining h*(TTL*f)-1.

In further embodiments of the present invention, the cache includes a disk cache and caching the data includes offloading cached memory contents to the disk cache.

In still other embodiments of the present invention, determining a predicted maximum number of cache accesses includes monitoring cache accesses to determine an update rate of cache entries corresponding to the distributed data. Determining a time (h) to process a cache hit corresponding to the distributed data may include monitoring cache accesses to determine the time (h). Determining a time (r) to replicate the distributed data may include monitoring data replication operations to determine the time (r). Finally, determining time (c) to generate the distributed data may include monitoring generation of the distributed data to determine the time (c).

As will further be appreciated by those of skill in the art, while described above primarily with reference to method aspects, the present invention may be embodied as methods, apparatus/systems and/or computer program products.

Brief Description of the Drawings

Figure 1 is a block diagram of a conventional application server cluster utilizing data replication;

Figure 2 is a block diagram of a conventional application server cluster utilizing disk caching;

Figure 3 is a block diagram of a data processing system suitable for use in cache controlling systems according to embodiments of the present invention;

Figure 4 is a more detailed block diagram of a system incorporating cache controlling according to embodiments of the present invention;

Figure 5 is a flowchart illustrating operations for cache control according to embodiments of the present invention; and

Figure 6 is a flowchart illustrating operations for cache control according to further embodiments of the present invention.

Detailed Description of the Invention

The present invention now will be described more fully hereinafter with reference to the accompanying drawings, in which illustrative embodiments of the invention are shown. This invention may, however, be embodied in many different forms and should not be construed as limited to the embodiments set

forth herein; rather, these embodiments are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the invention to those skilled in the art. Like numbers refer to like elements throughout.

As will be appreciated by one of skill in the art, the present invention may be embodied as a method, data processing system, or computer program product. Accordingly, the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects all generally referred to herein as a "circuit" or "module." Furthermore, the present invention may take the form of a computer program product on a computer-usable storage medium having computer-usable program code embodied in the medium. Any suitable computer readable medium may be utilized including hard disks, CD-ROMs, optical storage devices, a transmission media such as those supporting the Internet or an intranet, or magnetic storage devices.

Computer program code for carrying out operations of the present invention may be written in an object oriented programming language such as Java®, Smalltalk or C++. However, the computer program code for carrying out operations of the present invention may also be written in conventional procedural programming languages, such as the "C" programming language. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer. In the latter scenario, the remote computer may be connected to the user's computer through a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

The present invention is described below with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program

instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer-readable memory that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer-readable memory produce an article of manufacture including instruction means which implement the function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide steps for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

Various embodiments of the present invention will now be described with reference to the figures. Embodiments of the present invention may be incorporated into the conventional systems for replication and/or offloading of data discussed above with reference to **Figures 1** and **2**. However, embodiments of the present invention may also be utilized in other systems and with other configurations where decisions on whether to cache data may be made based on an assessment of the efficiency of caching the data. Thus, the present invention should not be construed as limited to use in systems such as those illustrated in **Figures 1** and **2** but may be used in any system utilizing distributed data. As used herein, distributed data is data that may either be cached for future use or re-generated for future. Embodiments of the present invention may dynamically determine if caching or re-generation may be more efficient and, thereby, control whether and/or where to cache (in a memory cache and/or disk cache) the data.

Figure 3 illustrates an exemplary embodiment of a data processing system **130** suitable for providing cache control in accordance with embodiments of the present invention. Such a data processing system may, for example, be provided as the MOM **14** and/or application servers **10** of **Figures 1** and **2**. The data processing system **130** of **Figure 3** is, thus, illustrative of data processing systems which may provide cache control, however, embodiments of the present invention may be provided by any type of data processing system capable of carrying out the operations described herein.

The data processing system **130** may include input device(s) **132** such as a keyboard or keypad, a display **134**, and a memory **136** that communicate with a processor **138.** The data processing system **130** may further include a speaker **144**, and an I/O data port(s) **146** that also communicate with the processor **138**. The I/O data ports **146** can be used to transfer information between the data processing system **130** and another computer system or a network. These components may be conventional components, such as those used in many conventional data processing systems, which may be configured to operate as described herein.

Figure 4 is a block diagram of data processing systems that illustrate systems, methods, and computer program products in accordance with embodiments of the present invention. The processor **138** communicates with the memory **136** via an address/data bus **248**. The processor **138** can be any commercially available or custom microprocessor. The memory **136** is representative of the overall hierarchy of memory devices containing the software and data used to implement the functionality of the data processing system **130**. The memory **136** can include, but is not limited to, the following types of devices: cache, ROM, PROM, EPROM, EEPROM, flash memory, SRAM, and DRAM.

As shown in **Figure 4**, the memory **136** may include several categories of software and data used in the data processing system **130**: the operating system **252**; the application programs **254**; the input/output (I/O) device drivers **258**; and the data **256**. As will be appreciated by those of skill in the art, the operating system **252** may be any operating system suitable for use with a data processing system, such as OS/2, AIX or System390 from International Business Machines Corporation, Armonk, NY, Windows95, Windows98, Windows2000 or

WindowsXP from Microsoft Corporation, Redmond, WA, Unix or Linux. The I/O device drivers **258** typically include software routines accessed through the operating system **252** by the application programs **254** to communicate with devices such as the I/O data port(s) **146** and certain memory **136** components. The application programs **254** are illustrative of the programs that implement the various features of the data processing system **130** and preferably include at least one application which supports operations according to embodiments of the present invention. Finally, the data **256** represents the static and dynamic data used by the application programs **254**, the operating system **252**, the I/O device drivers **258**, and other software programs that may reside in the memory **136.**

As is further seen in **Figure 4**, the application programs **254** may include a cache control module **260**. The cache control module **260** may carry out the operations described herein for determining whether to cache data or not and/or where to cache. The data portion **256** of memory **136**, as shown in the embodiments of **Figure 4**, may, optionally, include cache control data **262**. The cache control data **262** may be utilized by the cache control module **260** to determine whether caching or re-generation of the data is more efficient and, thereby, dynamically control the caching decision. The cache control data **262** also may be utilized by the cache control module **260** to determine where to cache (for example, to memory, disk, or replicated in the cluster)

While the present invention is illustrated, for example, with reference to the cache control module **260** being an application program in **Figure 4**, as will be appreciated by those of skill in the art, other configurations may also be utilized while still benefitting from the teachings of the present invention. For example, the cache control module **260** may also be incorporated into the operating system **252,** the I/O device drivers **258** or other such logical division of the data processing system **130**. Thus, the present invention should not be construed as limited to the configuration of **Figure 4** but is intended to encompass any configuration capable of carrying out the operations described herein.

Operations according to embodiments of the present invention will now be described with reference to the flowcharts of **Figures 5** and **6**. The operations of **Figures 5** and/or **6** may be carried out, for example, by the cache control module

260 of **Figure 4**. As seen in **Figure 5**, characteristics of the data to be cached are determined (block **500**). Such characteristics may include, for example, the time to live (TTL) of the data and the frequency (f) with which the data is accessed Such characteristics may be dynamically determined or pre-established. In particular embodiments of the present invention, the TTL information may be computed directly based on user input and/or determined by observing and recording (monitoring) the update rate for a particular cache entry. The frequency of cache access could be calculated and/or sampled based on the access pattern for a particular item or type of cache data.

Characteristics of the source of the data to be cached are also determined (block **502**). Such characteristics may include, for example, how long it takes to recompute the data. Such a determination may be dynamically made based on monitoring of the generation of the data. Characteristics of the cache are also determined (block **504**). Such characteristics may include, for example, how long it takes to replicate/offload the data and/or how long it takes to retrieve a cached item. These characteristics may be determined, for example, by sampling or otherwise monitoring the system as replication/offload of data and/or cache retrievals are performed so that the characteristics may be updated as conditions change.

The determined characteristics are evaluated (block **506**) to determine if caching the data is more efficient than re-generating the data, and if so, where to cache the data. For example, the decision to cache or not to cache may utilize the system-load statistics, frequency and response times to determine the configuration of a distributed cache. Based on the evaluation of the information described above, the decision may be made. For example, if the data can quickly be computed by a single node in the cluster, then it might be more efficient not to distribute the data. If the system is experiencing peak loads or if the data being replicated is large, causing delays in replication, replication might not be the most effective way to process the data. A similar determination could be made with respect to offloading the data to a disk cache. Also, if the data is updated frequently then the regeneration of the data may be more effective than distributing the data. Thus, replication may not be productive. Finally, retrieving data from the cache,

typically, takes time to calculate a cache key as well as physically retrieve the item from the cache. Thus, if the cache latency is too great it may be more efficient to distribute the data through caching.

If caching is more efficient (block **508**) the data is cached (block **510**), for example, by setting an indication that the data should be cached and using the indication to cache the data using conventional techniques. If caching is not more efficient (block **508**) the data is not cached (block **512**), for example, by setting the indication that the data should not be cached and using the indication to not cache the data, using conventional techniques.

Figure 6 illustrates operations for controlling a cache according to further embodiments of the present invention such as may be carried out, for example, by the cache control module **260** of **Figure 4**. As seen in **Figure 6**, a predicted time (c) it takes to generate the data to be cached is determined (block **600**). Such a predication may be made, for example, by monitoring the generation of the data and utilizing that information to determine the predicted time (c). Other techniques for determining the time (c) may also be utilized. This data may, for example, be stored as the cache control data **262** of **Figure 4**.

A predicted time (r) to replicate and/or offload the data to disk is also determined (block **602**). Such a predication may be made, for example, by monitoring the replication and/or offloading of the data and utilizing that information to determine the predicted time (r). Other techniques for determining the time (r) may also be utilized. This data may, for example, be stored as the cache control data **262** of **Figure 4**. Accordingly, a determination whether and/or where to cache may be made.

A time to live (TTL) of the data is also determined (block **604**). Such a determination may be made, for example, by computing the TTL directly based on user input and/or by monitoring the update rate for a particular cache entry. Thus, a projected TTL may be determined, for example, utilizing the average update timing for a cache entry corresponding to the data. Other techniques for determining TTL for a cache entry may also be utilized. This data may, for example, be stored as the cache control data **262** of **Figure 4**.

A time (h) to process a cache hit is also determined (block **606**). Such a determination may be made, for example, by monitoring cache access times. The monitored cache access times could be access times in general or access times for a cache entry corresponding to the data. Thus, a projected time (h) may be determined, for example, utilizing the average cache access time or the actual or average cache access time for a cache entry corresponding to the data. Other techniques for determining the time (h) may also be utilized. This data may, for example, be stored as the cache control data **262** of **Figure 4**.

A frequency (f) of cache accesses is also determined (block **608**). Such a determination may be made, for example, by calculation and/or sampling based on the access pattern to the particular item or type of cached data. Other techniques for determining the time (f) may also be utilized. This data may, for example, be stored as the cache control data **262** of **Figure 4**.

The values determined above are used to determine whether to cache the data, replicate the cached data in the cluster, or store the cached data to a disk. In particular, a determination is made as to whether c+r+(h*((TTL*f)-1)) is less than c*(TTL*f) (block **610**). This determination may be made repeatedly using the value of r that is equal to the time to retrieve the data from the local cache, the time to replicate the data in the cluster, or the time to offload the cache item to disk, to thereby determine whether and/or where to cache. If so, the data is cached (block **612**) and if not, the data is not cached (block **614**). As further explanation, the determination of the product of the frequency of access and the time to live of the cache entry (TTL*f) determines a predicted number of cache accesses during the life of a cache entry. The total time to process cache hits is given by h*((TTL*f)-1). Thus, the total time to obtain content from the cache is provided by the sum of the time to generate the data, the time to replicate the data and the time to process cache hits, which may be expressed as c+r+(h*((TTL*f)-1)).

The total time to regenerate the data if it is not cached is provided by the product of the time to generate the data and the predicted number of accesses of the data, which is c*(TTL*f). If the total time to regenerate the data is greater than the total time to obtain the content from the cache, then the data may be more efficiently cached. Otherwise, the data may be more efficiently regenerated.

As an example, if for data TTL is 10 seconds, the time (c) to generate the data is 0.3 seconds, the frequency (f) of access is 3 hits per second, the time (r) to replicate or offload the data os 1 second and the time (h) to process a cache hit is 0.1 seconds, then c+r+(h*((TTL*f)-1)) is 4.2. The value for c*(TTL*f) is 9. Accordingly, the data would be cached.

The flowcharts and block diagrams of **Figures 1** through **6** illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products for autonomic cache control according to various embodiments of the present invention. In this regard, each block in the flow charts or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the blocks may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be understood that each block of the block diagrams and/or flowchart illustrations, and combinations of blocks in the block diagrams and/or flowchart illustrations, can be implemented by special purpose hardware-based systems which perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

In the drawings and specification, there have been disclosed typical illustrative embodiments of the invention and, although specific terms are employed, they are used in a generic and descriptive sense only and not for purposes of limitation, the scope of the invention being set forth in the following claims.